RANDOM NUMBERS FROM SMALL CALCULATORS

Donald W. Rankin
Army Materiel Test and Evaluation Directorate
US Army White Sands Missile Range
White Sands Missile Range, New Mexico 88002

ABSTRACT. Random number generators are notoriously wasteful of digits; however, applying an augmented precision technique to a linear congruential generator enables one to produce on even a small calculator a set of pseudo-random numbers which contains a useful number of elements. This paper sets forth such a method.

1. **INTRODUCTION.** Most modern computers and many programmable calculators include in their softwares a function for generating "random" numbers. Such numbers are required any time a "Monte Carlo" test technique is employed.

It is usual to tailor each algorithm to a specific type of use, and to a specific size of computer. Probably it is not feasible to transfer such a tailored algorithm to a calculator of smaller size--particularly to one of shorter word length.

Perhaps the most efficient and certainly the most popular of these algorithms is the "Linear Congruential Generator." Mathematically stated,

$$x_{i+1} \equiv (ax_i + c) \bmod m.$$

All quantities are considered to be integers. If the modulus m be taken as some power of ten (or of two if in binary), the modular operation is effected by simple truncation.

Most calculators have the ability to truncate at the decimal point. A decimal point, therefore, is inserted solely for this purpose. Conceptually, the numbers remain integers.

Given that the modulus m is some (positive integer) power of ten, it is found that the algorithm generates a full set of m integers (ranging from zero to m-1, inclusive) whenever both

$$a \equiv 1 \pmod{20} \text{ and}$$
$$c = (1, 3, 7, \text{ or } 9) \pmod{10}.$$

The selection of values for a and c is an important part of adapting the algorithm to a specific case.

2. PSEUDORANDOM NUMBERS. Let us suppose that we have defined a set of m integers, all different. A random selection from the elements of this set requires that for any element, the probability of selection be $1/m$. Since this probability remains unchanged for subsequent selections, sampling with replacement is indicated.

We wish to develop an algorithm that does not depend upon an outside stimulus. However, it remains necessary to provide a value for x_0, so that the process can begin. This value should be an element of the set, but the choice can be arbitrary. It is called the "seed." After each x_j is computed and used, it serves in turn as the "seed" for the next calculation. To avoid repetition, some programmers employ a date-time group from which to extract a value for x_0.

If any computed value of

$$x_{j+s+1} \equiv (ax_{j+s} + c) \pmod{m}$$

is ever equal to some previously used value of x_j, the algorithm will repeat itself over a subset of size (s+1), exactly duplicating the previous cycle. If $x_{j+1} = x_j$, it is found that $s = 0$, and the algorithm has already degenerated into uselessness. To circumvent this, sampling without replacement is used. But this causes the probability of selection to increase as

$$\frac{1}{s}, \frac{1}{s-1}, \frac{1}{s-2}, \ldots, \frac{1}{s-(s-1)}.$$

Thus, the last remaining element of the subset can be predicted with certainty. It is, of course, equal to x_j, the seed which began the cycle.

How then can we presume to use these sequences of numbers as "random" sequences? It is found that if the cycle length is very large (two hundred-fold would not be excessive) when compared with the quantity (of numbers) required, the sequence selected will exhibit certain of the characteristics associated with random sequences.

The term "pseudorandom" is used to indicate that the sequence is generated by an algorithm so that each element is a function of its predecessor.

3. PARAMETER SELECTION. At this point, let us limit the discussion to the case

$$m = 10^{2e},$$

"e" being a small, positive integer. Immediately

$$\sqrt{m} = 10^{e}.$$

It was observed in Section 1 that, under these conditions, maximum cycle length is achieved if c and m are relatively prime, and additionally $a \equiv 1 \pmod{20}$.

There are other requirements, however. Foremost among these is the restriction that ax_j must never overflow the computer word length. Should this occur, digits will be lost from the right, interrupting the flow of the algorithm and seriously shortening the cycle length.

The formula for serial correlation is

$$\rho = \frac{1 - 6\left(\frac{c}{m}\right)\left(1 - \frac{c}{m}\right)}{a} + \varepsilon$$

where $|\varepsilon| < \frac{a}{m}$.

It can be seen that the numerator varies from -0.5 to +1.0 and that the two terms are of the same order of magnitude when

$$a^2 \approx m.$$

The numerator can be reduced to zero by solving the associated quadratic in c/m. It is found that

$$\frac{c}{m} = \frac{1}{2} \pm \frac{1}{6}\sqrt{3.}$$

Now $\frac{1}{6}\sqrt{3}$ = 0.28867 51345 94812 88225 45743 90 . . . is irrational, so that no element of the set can furnish a value for c which will reduce the numerator exactly to zero. It can, however, be made quite small, whence "a" can be set to a value somewhat less than $\sqrt{m}$ without adversely affecting the serial correlation.

At this point, it will be instructive to examine the sequence generated by the following parameters:*

$$x_0 = 0$$
$$a = 81$$
$$c = 788677$$
$$m = 1000000$$

This sequence is found in Table 1-1. The entries are to be read as integers.

It is easy to observe that the least significant digit (units digit) is not "random" at all, since it can be predicted exactly. In the case at hand,

*All examples in this paper will assume an 8-digit calculator.

TABLE 1-1.

CYCLES OF DIGITS

$a = 81$ $c = 708677$

$m = 1{,}000{,}000$ $x_0 = 0$

0.708677	0.634957	0.137237
0.671514	0.220194	0.904874
0.181311	0.624391	0.083471
0.474868	0.364348	0.549828
0.252985	0.300865	0.324745
0.280462	0.158742	0.093022
0.506099	0.646779	0.323459
0.782696	0.177776	0.988856
0.187053	0.188533	0.886013
0.939970	0.059350	0.555730
0.926847	0.636527	0.302807
0.814684	0.347364	0.816044
0.778081	0.925161	0.888241
0.813238	0.726718	0.736198
0.660955	0.652835	0.420715
0.326032	0.668312	0.866592
0.197269	0.921949	0.982629
0.767466	0.466546	0.381626
0.953423	0.578903	0.700383
0.915940	0.679820	0.519700
0.079817	0.854097	0.884377
0.253854	0.970534	0.423214
0.350851	0.401931	0.069011
0.207608	0.345088	0.378568
0.604925	0.740805	0.452685
0.787602	0.793882	0.456162
0.584439	0.093119	0.737799
0.128236	0.331316	0.550396
0.175793	0.625273	0.370753
0.027910	0.435790	0.819670
0.019387	0.087667	0.181947
0.789024	0.889704	0.926384
0.699621	0.854701	0.426781
0.457978	0.019458	0.276938
0.884895	0.364775	0.220655
0.465172	0.335452	0.661732
0.467609	0.960289	0.388969
0.665006	0.572086	0.295166
0.654163	0.127643	0.697123
0.775880	0.127760	0.255640

it cycles through all ten digits, then repeats itself exactly. The two least significant digits, viewed as a single number, exhibit a similar cycle. A good generator will continue this effect until a cycle of length m is achieved. If m is a power of 10, this maximum cycle length is obtained whenever <u>both</u> of the following conditions are met:

1. $a \equiv 1 \pmod{20}$

2. c and m are relatively prime. This requires only that the final (units) digit of c be 1, 3, 7, or 9.

As an aid to continuing the study of the cycling effect, let us define

$$a_s = a^s \pmod{m}$$

and

$$c_s = c\,(1 + a + a^2 + \ldots + a^{s-1}) \pmod{m}.$$

Given $a = 81$, $s = 10$, $c = 788677$ we find

$$a_{10} = 928801$$
$$c_{10} = 939970$$

Note that, since $x_0 = 0$, c_{10} appears in the tenth position in Table 1-1. Now c_{10} may be viewed as having only five digits. It is therefore completely exercized by a five-digit multiplier, and we need merely use the last five digits of a_{10}. The parameters

$$x_0 = 0$$
$$a_{10} = 28801$$
$$c_{10} = 93997$$
$$m_{10} = 100\,000$$

will generate the sequence $x_{10}, x_{20}, x_{30}, \ldots$.

At first glance, this appears to exceed the calculator word length. However, if we multiply x_i $(a_{10}-1)$ and then truncate, the algorithm will run without difficulty. To express the complete formula

$$x_{i+10} \equiv x_i (a_{10} - 1) \pmod{m} + x_i + c_{10} \pmod{m}$$

It is convenient to compute a_s by means of the binomial expansion. Hence

$$\begin{aligned}(1 + 80)^{10} &= 1 + 10(80) + 45(6400) + \\ &+ 120(80)^3 + 210(80)^4 + \\ &+ 252(80)^5 + \text{immaterial terms}\end{aligned}$$

The previous strategem will thus be available whenever s is a multiple of ten. The sequence thus generated is found in Table 1-10.

In a similar manner, the procedure can be reiterated and the sequence $x_{100}, x_{200}, x_{300}, \ldots$ generated. Required values of the parameters are:

$$\begin{aligned}x_0 &= 0 \\ a_{100} &= 8001 \\ c_{100} &= 5197 \\ m_{100} &= 10\,000.\end{aligned}$$

This sequence is illustrated in Table 1-100.

The process can be carried no farther. To do so results in $a_{1000} = 1$, and the algorithm degenerates to the successive multiples of x_{1000}. This can be observed by looking at every tenth entry in Table 1-100. The phenomenon can be called a "quasi-cycle" of length 1000 and additive constant 197. It appears that original values of "a" congruent to 1 (mod 100) will hasten this effect and therefore should be avoided. Further scrutiny reveals that the "quasi-cycle" is actually of length 500 and additive constant 598.5.

TABLE 1-10.

CYCLES OF DIGITS

a_{10} = 28801 c_{10} = 93997

m = 100,000 x_0 = 0

0.939970	0.058770	0.777570
0.015940	0.574740	0.733540
0.027910	0.026710	0.625510
0.775380	0.214680	0.253480
0.059850	0.938650	0.417450
0.679820	0.990620	0.917420
0.435790	0.194590	0.553390
0.127760	0.326560	0.125360
0.555730	0.194530	0.433330
0.519700	0.598500	0.277300
0.819670	0.338470	0.457270
0.255640	0.214440	0.773240
0.627610	0.026410	0.025210
0.735580	0.574380	0.013180
0.379550	0.658350	0.537150
0.359520	0.078320	0.397120
0.475490	0.634290	0.393090
0.527460	0.126260	0.325060
0.315430	0.354230	0.993030
0.639400	0.118200	0.197000
0.299370	0.218170	0.736970
0.095340	0.454140	0.412940
0.827310	0.626110	0.024910
0.295280	0.534080	0.372880
0.299250	0.978050	0.256850
0.639220	0.758020	0.476820
0.115190	0.673990	0.832790
0.527160	0.525960	0.124760
0.675130	0.113930	0.152730
0.359100	0.237900	0.716700
0.379070	0.697870	0.616670
0.535340	0.293040	0.032640
0.627010	0.825810	0.624610
0.454980	0.093780	0.332580
0.818950	0.897750	0.576550
0.518920	0.037720	0.156520
0.354890	0.313690	0.872490
0.126860	0.525660	0.524460
0.634830	0.473630	0.912430
0.678800	0.957600	0.836400

TABLE 1-100.

CYCLES OF DIGITS

$a_{100} = 8001$ $c_{100} = 5197$

$m_{100} = 10,000$ $x_0 = 0$

0.519700	0.307700	0.095700
0.639400	0.427400	0.215400
0.359100	0.147100	0.935100
0.678800	0.466800	0.254800
0.598500	0.386500	0.174500
0.118200	0.906200	0.694200
0.237900	0.025900	0.813900
0.957600	0.745600	0.533600
0.277300	0.065300	0.853300
0.197000	0.985000	0.773000
0.716700		
0.836400	0.504700	0.292700
0.556100	0.624400	0.412400
0.875800	9.344100	0.132100
0.795500	0.663800	0.451800
0.315200	0.583500	0.371500
0.434900	0.103200	0.891200
0.154600	0.222900	0.010900
0.474300	0.942600	0.730600
0.394000	0.262300	0.050300
	0.182000	0.970000
0.913700		
0.033400	0.701700	0.489700
0.753100	0.821400	0.609400
0.072800	0.541100	0.329100
0.992500	0.860800	0.648800
0.512200	0.780500	0.568500
0.631900	0.300200	0.088200
0.351600	0.419900	0.207900
0.671300	0.139600	0.927600
0.591000	0.459300	0.247300
	0.379000	0.167000
0.110700		
0.230400	0.898700	0.686700
0.950100	0.018400	0.806400
0.269800	0.738100	0.526100
0.189500	0.057800	0.845800
0.709200	0.977500	0.765500
0.828900	0.497200	0.285200
0.548600	0.616900	0.404900
0.868300	0.336600	0.124600
0.788000	0.656300	0.444300
	0.576000	0.364000

The conclusion to be drawn is this: Even though the values of a and c be chosen so that the algorithm generates the full cycle of m integers before repeating, the number of elements of any "useful" subset probably does not exceed $\frac{1}{2}\sqrt{m}$. What is needed is a device to increase the effective word length of the calculator. How this can be done forms the subject matter of the next section.

In summary, let us view the number $ax_i + c$ before truncation. Obviously, the left-hand (most significant) digits are lost via the modular operation, leaving

$$x_{i+1} \equiv (ax_i + c)(\text{mod } m).$$

Now the x_i can assume, at most, "m" different values. Therefore, since both "a" and "c" are fixed, the quantity $ax_i + c$ also can assume, at most, "m" different values. What this means is that, provided the values of "a" and "c" are selected to produce maximum cycle length, the act of truncation does not reduce the quantity of numbers--only their size. It also shuffles their order.

What remains is, of course, x_{i+1}. It is usual to regard several of the right-hand (least significant) digits as "not significantly random." They are retained, however, for smooth operation of the algorithm, and to ensure that the full complement of "m" different numbers is delivered.

4. AUGMENTED PRECISION ARITHMETIC.

Double precision arithmetic is available in the software of many computers, and even in some calculators. It is cumbersome to program and executes very slowly. This is particularly true with division.

However, the algorithm for the linear congruential generator does not employ division. Moreover, since $a^2 < m$, the word length $(m\sqrt{m} - 1)$ is sufficient.

Let $m = 10^{2e}$, where e is any small positive integer. The "augmented" word consists of three parts, each of which consists of "e" digits.

Let us express x_i in the form

$$x_i = u_i \times 10^e + v_i$$

Thus a, u_i, and v_i are all integers less than $\sqrt{m}$, and the product of any two of them will not cause overflow.

Some calculators will compute (but not necessarily display) an extra digit. For them, the procedure is extremely easy. First, compute

$$(au_i \times 10^e) \pmod m.$$

To this quantity, add $(av_i + c)$ and truncate again. The result is x_{i+1}.

When place for an extra digit is lacking, it is necessary to devise a procedure which avoids overflow. The following method, which assembles x_{i+1} by parts, beginning at the right, works quite well.

As before, express x_i in the form

$$x_i = u_i \times 10^e + v_i.$$

In analogous fashion, express "c" as

$$c = p \times 10^e + q,$$

Store p, q, u_i, and v_i separately. Select "a" so that

$$a < 10^e$$
$$a \equiv 1 \pmod{20}$$
$$a \not\equiv 1 \pmod{100}$$

It will be found that $a < 10^e - 18$. Consequently, multiplication by parts will not produce overflow.

We have immediately

$$v_{i+1} \equiv (av_i + q) \bmod 10^e.$$

Since we wish to retain both parts of $(av_i + q)$, we compute $(av_i + q) \times 10^{-e}$, then "FRC" $((av_i + q) \times 10^{-e})$.*

v_{i+1} is now stored, replacing v_i. Then $u_{i+1} \equiv (au_i + p + 10^e(av_i + q - v_{i+1})) \bmod 10^e$.

The sequence of numbers generated by

$$x_0 = 0$$
$$a = 9941$$
$$c = 2113\ 2487$$
$$m = 100\ 000\ 000$$

is displayed in Table 2-1.

5. RANDOM SELECTION. RANDOM ORDERING.

So far, an algorithm has been developed which will generate a full set of m pseudorandom numbers. However, the length of a useful sequence of these numbers is, at best, uncertain and doubtless does not exceed $\frac{1}{2}\sqrt{m}$.

If a subset of far smaller but exactly known size is to be placed in random order, or if random selections from its elements are to be made, the following can be done.

---

*"FRC" means "fractional part of."

Storage space must be provided to accommodate all the elements of the subset, plus one more. It may be possible that scratch-pad storage is adequate.

Let us illustrate the method by example. Suppose the task at hand is to shuffle a pack of 52 playing cards, i.e., to place them in random order. We thus require 53 storage registers, which we number from 00 to 52, inclusive. The individual card names are entered into registers 00 through 51 in any arbitrary order. N = 52 is the subset size.

We employ the generated sequence of numbers given in Table 2-1. These numbers (integers) should be distributed uniformly on the interval 0 to m. Dividing by m, then multiplying by 52, yields a sequence uniformly distributed on the interval 0 to 51.99999 . . . . The "integer" portion of this number is used as an address for selecting a card. That card is then placed in storage register 52.

Next, all cards with location numbers greater than the "selected" location are cascaded downward one position. This includes the card placed in register 52. So far, the illustrative example has given 52 x 0.21132487 = 10.988 . . . . The card in location 10 was drawn and stored in location 52. Say it is the Spade Jack.

After cascading, only 51 cards are of interest. Hence 51 × 0.99185754 = 50.584 . . . . The card in location 50--the King of Clubs--is drawn and placed in register 52. Again after cascading, the subset of unshuffled cards is reduced to 50 in number. Hence 50 × 0.26713001 = 13.356 . . . . The card now in location 13--the deuce of Hearts--is drawn and placed in register 52.

Continuing as above, 49 × 0.75075428 = 36.786 . . . . The card in location 36--say the King of Diamonds--is selected and placed in location 52.

When the size of the unshuffled subset is reduced to unity, that card certainly will be found in location 00, and it certainly will be selected for transfer to location 52. Consequently, that transfer can be effected without

TABLE 2-1.

CYCLES OF DIGITS

$a = 9941$ $c = 2113\ 2487$

$m = 100,000,000$ $x_0 = 0$

0.21132487	0.24103567	0.22722647
0.99185754	0.34692034	0.06966314
0.26713001	0.94642481	0.73259961
0.75075428	0.62036108	0.98404788
0.45962235	0.22082115	0.63129995
0.31710622	0.39437702	0.96412782
0.56425789	0.71328069	0.60598349
0.49900936	0.93466416	0.29319896
0.86337263	0.70773943	0.90218623
0.99863970	0.84899850	0.84463730
0.68858257	0.10541337	0.75072417
0.41065324	0.12563604	0.16029884
0.51518371	0.15919851	0.74209331
0,65258598	0.80371278	0.36091958
0.56855205	0.92007085	0.11286965
0.18725392	0.63564472	0.24851552
0.70254359	0.15548639	0.70410919
0.19715306	0.90152786	0.76078266
0.10989433	0.29978113	0.15174793
0.67085940	0.33553820	0.73749700
0.22462027	0.79657107	0.66900187
0.16142894	0.92433174	0.75891454
0.97641741	0.99315221	0.58076701
0.77679768	0.13744448	0.61617128
0.35706175	0.54690055	0.57001935
0.76218162	0.94969242	0.77368322
0.05880929	0.10367209	0.39621489
0.83447676	0.81557156	0.98354636
0.74479603	0.80820283	0.64568963
0.22865910	0.55565790	0.01193670
0.31143797	0.00650877	0.87405957
0.21618464	0.91500744	0.23751024
0.30283111	0.30028591	0.30062071
0.65538938	0.35355618	0.68180298
0.43715145	0.91331025	0.01474905
0.93388932	0.42852012	0.83163092
0.00505499	0.12983779	0.45430059
0.46298046	0.92879526	0.41349006
0.70007773	0.36500453	0.71601133
0.68403880	0.72135760	0.07995640

employing the algorithm. Further, cascading can be omitted or not, at the pleasure of the programmer.

The final result is the shuffled deck, in order of selection, in the designated storage locations. Omitting the final cascading, the example leaves the Spade Jack in 01, the Club King in 02, the Heart deuce in 03, the Diamond King in 04, etc. The shuffled deck can now be put to the use for which it was intended.

If there is a requirement to "deal" the cards one at a time, it is suggested that the card in the highest numbered location be taken first. Not only is the programming simpler, but the stigma is avoided which usually is attached to dealing from the bottom.

In summary, a set of uncertain size has been used to produce a much smaller subset of known, fixed size.

6. STATISTICAL TESTS. There is much to be found in the literature on the subject of testing sequences of numbers to determine whether or not a sequence could have been produced by a random selection process. These methods will not be repeated here.

It is enough to be reminded that the answers to these statistical tests will be stated as probabilities. We should read nothing into the result beyond the probability statement itself.

BIBLIOGRAPHY

(1) Abramowitz, M. and Stegun, I.A., eds., HANDBOOK OF MATHEMATICAL FUNCTIONS; Dover Publications, Inc., New York, 1972. (Sec 26.8)

(2) Hull, J. E. and Dobell, A.R., RANDOM NUMBER GENERATORS; in SIAM Review, Vol 4 No 3 July 1962.

(3) Jansson, B., RANDOM NUMBER GENERATORS; Almquist and Wiksell, Stockholm, 1966.